Cyclopath Route Finding White Paper – Cost Functions

Writ by L. Bouma / Last updated: 2011-12-02
w/ ackgt. to R. Priedhorsky , L. Terveen,
and the Cyclopath Cuttin' Crew.

ABSTRACT

This document describes how the Cyclopath Route Finders' cost functions work.

OVERVIEW

To find the best bicycling route between two points on a map, Cyclopath uses the A* search algorithm[1]. A* searches the Cyclopath road network to calculate the cheapest path between two points, or terminal nodes. It uses a best-first approach and requires us to design a distance-plus-cost heuristic which allows it to examine a subset of the road network to find the best path. The algorithm uses the geometry and attribute of each street segment and some user input to assign each segment and each intermediate node a traversal cost. The path with the least total cost is considered the best bicycling route for the user. This document focuses on how the costs are calculated, explaining other details as necessary.

BACKGROUND

Cyclopath uses one of two different sets of cost functions based on the user's input.

If the user is requesting a bike-only route, the A* algorithm uses the original cost functions, writ by R. Priedhorsky circa 2006-9. The original functions use user input, Cyclopath user ratings, and a modified version of the Chicagoland Bicycle Federation Map Criteria 7 (CBFMC^7)[2] to determine costs.

1 Wiki article on A* search: http://en.wikipedia.org/wiki/A*_search_algorithm

2 Chicago Bicycle Federation Map Criteria 7: http://bikelib.org/roads/roadnet.htm [broken link]

If the user is requesting a multimodal route, the A* algorithm uses a set of newer cost functions, writ by L. Bouma with help from C. Drysdale and lots of sample code from the folks who developed GraphServer[3].

There are a number of differences between the two sets of cost functions, but one major difference is the units that the costs represent. The original cost functions return values that express distance in meters. But in the newer cost functions for multimodal, the costs express time in seconds. The reason for the difference is that the multimodal route finder has to compare costs against the public transit network, and the cost of traversing transit edges and nodes is expressed in time[4].

Cyclopath Route Finder Cost Functions, V1

FROM THE USER'S PERSPECTIVE

The heart of the Cyclopath route finder is the Bikeability slider. The user is allowed to choose whether they'd like the shortest distance path, the “most bikeable” path, or some combination of the two.

The position of the slider influences how the A* algorithm computes the edge cost.

If the slider is “all the way left,” indicating Minimize Distance, the cost of an edge is simply its street segment's real-world length, expressed in meters. The path with the least cost is the path with the least real-world distance.

3 GraphServer: http://graphserver.github.com/graphserver/

4 See Google Transit Feed Specification: https://code.google.com/transit/spec/transit_feed_specification.html

If the slider is “all the way right,” indicating Favor Bikeability, the cost of each edge is still the real-world length of the street segment, but it's scaled according to the “bikeability rating” of the edge. The bikeability rating is determined either using existing user ratings or using various attributes about the edge to calculate a generic rating value. The bikeability rating can also be influenced by on-the-fly user input, such as tag preferences.

If the slider is “somewhere in the middle,” the cost is determined by scaling and combining both the distance cost and the bikeability cost. You can think of the slider as indicating the percentage of each of the two costs you want figured into the final cost of the edge.

For example, consider the following slider, where the pointer is just right of center. If you notice, there are nine tick marks, which represent values between 0% and 100% inclusive.

Distance %:	100%	...	75	...	50	...	25	...	0
Bikeability %:	0%	...	25	...	50	...	75	...	100

In this example, the cost of an edge is 37.5% of the distance cost plus 62.5% of the bikeability cost.

DISTANCE COST FUNCTION

The distance cost function is pretty simple. It is calculated using the length of the line segments in the geodatabase, which correspond to the lengths of the associated street segments in the real world, as well as figuring in a little penalty for transitioning between edges.

For example, if a street segment is one mile long in the real world, its distance-cost is 1,609.344 (the number of meters in a mile).

But the distance cost function also considers the cost of transitioning edges. That is, where two road segments meet, there may be a cost associated with traveling from one edge to the next.

For example, consider the node where two perpendicular road segments intersect (i.e., a four-way intersection). Oftentimes, there are vehicles and traffic controls that make it more costly to transition from one road segment to the next. E.g., turning left generally takes longer than turning right because you have to cross traffic, or wait for a green arrow, etc.

For example, if a path follows a street segment that is 100 meters long, and then the next street segment requires a 90-degree left turn before continuing another 100 meters, and assuming the "penalty" for turning left is 10 meters, the total path cost is 100 + 10 + 100 = 210.

As such, the total distance cost includes not just the edge costs but also the edge transition (or node traversal) costs.

In graph jargon, the edge cost and the node traversal cost are called state transition costs. I.e., each edge and each node represents a "state," and the distance cost function just adds all of the costs of all transitions between states. I.e., from the origin node, to the first edge, to the second node, to the second edge, to the third node, etc., until the goal node is reached. This distinction will be more important when we discuss the multimodal route finder, which has a handful more than just two states.

FIXME: illustrate the node transition cost?

BIKEABILITY COST FUNCTION

The bikeability cost function is more complicated. It is calculated also using the lengths of the line segments in the geodatabase, but the length is scaled depending on the attributes of and ratings for the edge, as well as by on-the-fly user preferences.

I. Segment Ratings

Each street segment is assigned a generic rating based on the CBFMC^7.

Also, users can "rate" each street segment and assign their own personal rating.

A street segment rating is a value between 0 and 5. A higher value indicates a more bike-friendly road, while a lower value indicates a less bike-friendly road.

At the extremes, a 0-value is used to completely avoid a street segment (assign it an infinite cost), and a 5-value is used to absolutely preference a street segment (assign it a zero cost). In practice, you'll never see a 5-value (you'll see values approaching 5), but you will find 0-values on unbikeable streets (such as Controlled-access freeways).

Each generic or user rating has a value between 0 and 4. The values correspond to the following qualitative descriptions:

Excellent – 4
Good – 3
Fair – 2
Poor – 1
Impassable – 0
Don't Know – *(removes the rating)*

i. Deliberate Ratings (User-Applied)

Every street segment may be rated once by every user, using the scale described earlier.

ii. Generic Rating (System-Guessed)

Every street segment in the road network is initially assigned a “generic user rating,” which is really just a guess using the modified CBFMC^7 algorithm. This value is superseded by any user ratings, as described later.

The CBFMC^7 is only used for certain road classifications.

a. Controlled-Access and Non-Motorized Road Classes

If the segment is classified as a controlled-access road, such as a freeway, the rating is 0.

If the segment is classified as a bicycle path or sidewalk, it's rated based on its length.

> A bicycle path segment that is 500 meters or longer is rated 4 (Excellent). A path segment between 100 and 500 meters is rated 3.5 (Good to Excellent). And a short path segment, 100 meters or less, is rated 2.5 (Fair to Good).
>
> A sidewalk is similarly rated: 3 (Good) for a long sidewalk, greater than 500 meters, 2.5 (Fair to Good) for a medium-length sidewalk between 100 and 500 meters, and 1.5 (Poor to Fair) for short segments of 100 meters or less.

(Obviously, this setup is buggy. Consider the Greenway, for instance, which has many small segments because of on and off ramps. These ramps do not impede flow on the trail, but the segments of the trail are nonetheless rated incorrectly, e.g., rated 2.5 instead of 4.)

b. Motorized Road Classes

If the segment is otherwise a road that allows motor vehicle traffic, and not a bicycle path or a sidewalk, the generic rating algorithm examines the attributes of the street segment.

– Intermediate Rating

The CBFMC^7 uses three attributes: average annual daily traffic (AADT), speed limit, and lane count.

If the three attributes are not available, an different rating value is used. For highways, the rating is 1 (Poor). For Major Roads and segments classified as Unknown or Other, the segment is rated 2 (Fair). For all other segment classifications -- Local Road, Double track, and Single track -- the segment is rated 3 (Good).

(Again, this seems like a bad rating system: whose to say biking on the highway should be rated Poor? Some highways are really nice to bike on. And why would a bicycle trail that's a short segment get rated a 2.5, less than a local road?)

If the three attributes are available, they are used to lookup the intermediate rating using the CBFMC^7. The value, which depends on the shoulder width, amount of traffic, and number of lanes, is generally a 1, 2, or 3 (or Poor, Fair, or Good). For some segments, like busy 35-mph roads, or moderately busy 45-mph or 55-mph roads, the value is a strict zero, meaning avoid.

(FIXME: The CBFMC^7 should not return a 0, which translates to an avoid. What's currently 0 seems like more of a Poor rating.)

– *Final Rating*

Finally, the intermediate rating is penalized or promoted based on a few other attributes.

If the shoulder width is 4 feet or greater or if the street segment is tagged with 'bikelane', the intermediate rating is increased by 1.5 rating points.

If the street segment is tagged with 'bikelane' (regardless of the shoulder width), the intermediate rating is increased by ½ (so a shouldered road with a bike lane gets a 2 point boost).

Thirdly, if the segment is tagged unpaved, the rating is downgraded by 1 point.

(FIXME: I'm not sure that unpaved streets should be demoted. Shouldn't that be left up to the tag preferences to decide?)

II. Choose a Rating

The bikeability cost function uses the street segment rating to calculate the edge cost. Only one rating is used – either the generic rating, the user's rating, or an average of all user ratings.

If the user that initiated the route-finder request has rated the street segment, their rating for that segment is used for the cost function. If the user has not rated the segment, or if the user is not logged in, then the generic CBFMC^7 rating is used, unless one or more other users have rated that segment, in which case the average of all user ratings is used.

(Previous research has shown that the value of one or more user ratings is more accurate than the generic, calculated rating value [FIXME: citation needed], which is why we don't bother with the CBFMC^7 value if one or more user ratings apply to a street segment.)

Figure 1. shows a flow chart that illustrates choosing a rating for a street segment.

III. Calculate the Traversal Cost

Using the rating chosen for the street segment, the traversal cost can now be calculated.

To summarize the process:

- A street segment that's rated 0 or that has a tag that matches an “avoid” preference is given a infinite cost. This means the algorithm will not consider this street segment.
- The rating chosen in the previous step is averaged with the tag preference ratings.
  - For each street segment tag that matches a “bonus” preference, a tag preference rating of 5 is added to the sum of ratings.
  - For each street segment tag that matches a “penalty” preference, a rating of ½ is added to the ratings sum.
  - The mean of the sums of ratings is a value between ¾ and 4-1/2.
- The mean rating is inverted and scaled so we can use it to scale the street segment length.
  - The mean rating is divided by 5 – the maximum rating value – and subtracted from 1, which produces a number between 0.10 and 0.85.
  - This number is multiplied by 3 and then multiplied by the street segment length,

which generates a weighted length between 0.30 and 2.55 times the street segment length.

- Finally, the weighted length is combined with the unweighted length, according to the Bikeability slider.
  - Recall from the earlier discussion about the Bikeability slider that it indicates two values – a percentage of minimized distance and a percentage of maximized bikeability. The two values when added together equal one.
  - The final cost for the street segment is the percentage of minimized distance multiplied by the actual street segment length added to the percentage of maximized bikeability multiplied by the weighted street segment length.

The process of calculating the cost as just described is illustrated in Figure 2.

The following table illustrates how one tag preference interacts with a user's rating. A smaller weighted scalar means a segment is more bikeable. FIXME: Is 1.20 really the middle? Why?

tag preference	rating	averaged rating	weighted scalar	effect
thumbs down	'fair'	(0.5+2)/2 = 1.25	3*(1-1.25/5) = 2.25	> 1.20
thumbs down	'good'	(0.5+3)/2 = 1.75	3*(1-1.75/5) = 1.95	> 1.20
thumbs down	'excl'	(0.5+4)/2 = 2.25	3*(1-2.25/5) = 1.65	> 1.20
thumbs up	'fair'	(5.0+2)/2 = 3.50	3*(1-3.50/5) = 0.90	< 1.20
thumbs up	'good'	(5.0+3)/2 = 4.00	3*(1-4.00/5) = 0.60	< 1.20
thumbs up	'excl'	(5.0+4)/2 = 4.50	3*(1-4.50/5) = 0.30	< 1.20
avoid	'fair'	(0.0+2)/2 = 1.00	3*(1-1.00/5) = 2.40	> 1.20
avoid	'good'	(0.0+3)/2 = 1.50	3*(1-1.50/5) = 2.10	> 1.20
avoid	'excl'	(0.0+4)/2 = 2.00	3*(1-2.00/5) = 1.80	> 1.20
no tag pref	'fair'	(2)/1 = 2.00	3*(1-2.00/5) = 1.80	> 1.20
no tag pref	'good'	(3)/1 = 3.00	3*(1-3.00/5) = 1.20	= 1.20
no tag pref	'excl'	(4)/1 = 4.00	3*(1-4.00/5) = 0.60	< 1.20

NODE TRAVERSAL COST

The cost of transitioning from one edge to the next edge – traversing a node – depends on the change in direction of travel.

If the transition is akin to a left turn, an extra 100 meters is added to the path. For a right turn, 40 meters, and for going forward, 20 meters.

penalty left = 100 (meters)
penalty right = 40
penalty straight = 20

(FIXME: I don't agree with this heuristic, since it doesn't account for “true” intersections, i.e., those with traffic controls or those on streets and not where a trail entrance meets a trail. One good example is routing along the W River Road bicycle trail, where northbound bike traffic is unimpeded at node junctions.)

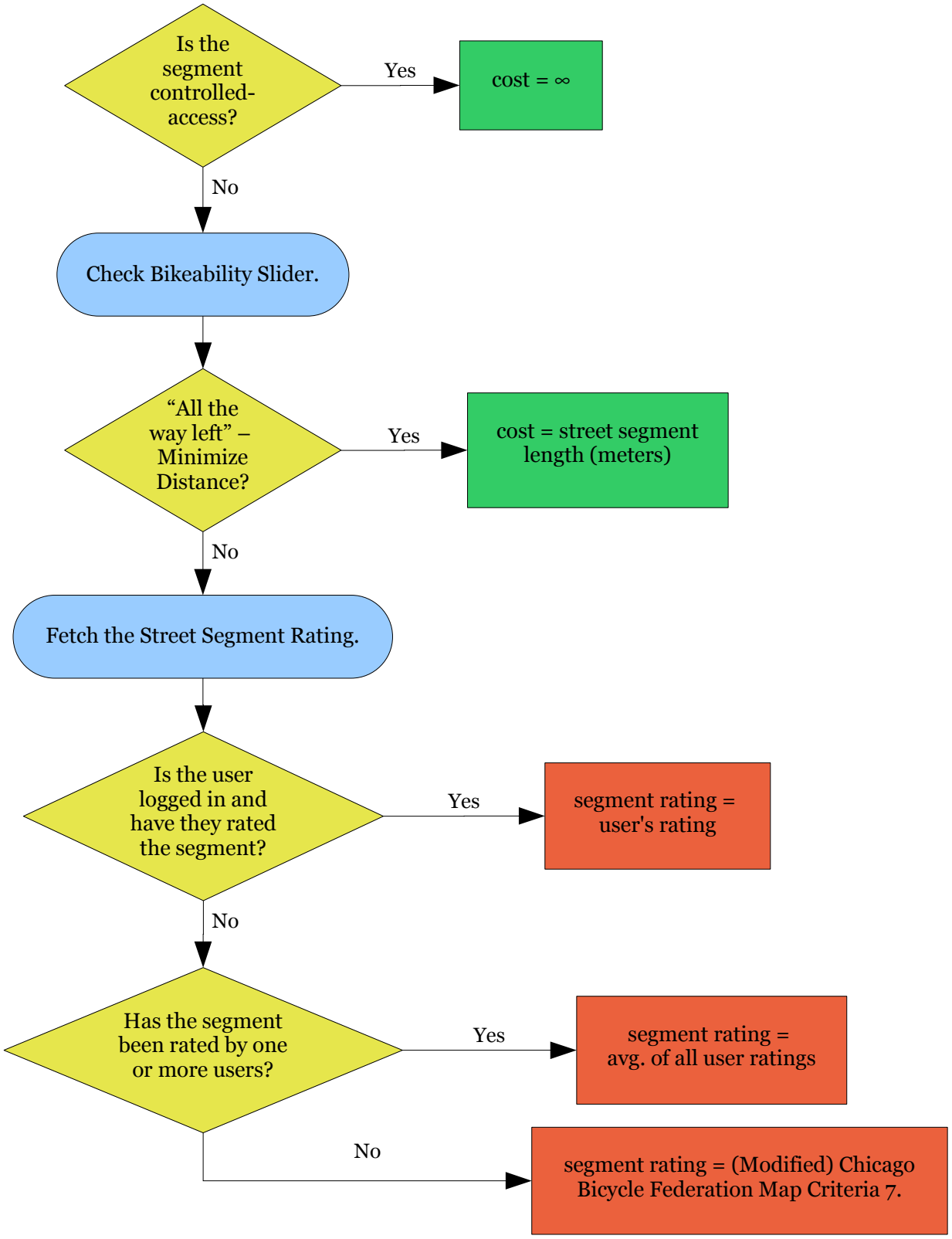


Figure 1. Cyclopath V1 Edge Traversal Cost Flow Chart (Part I).

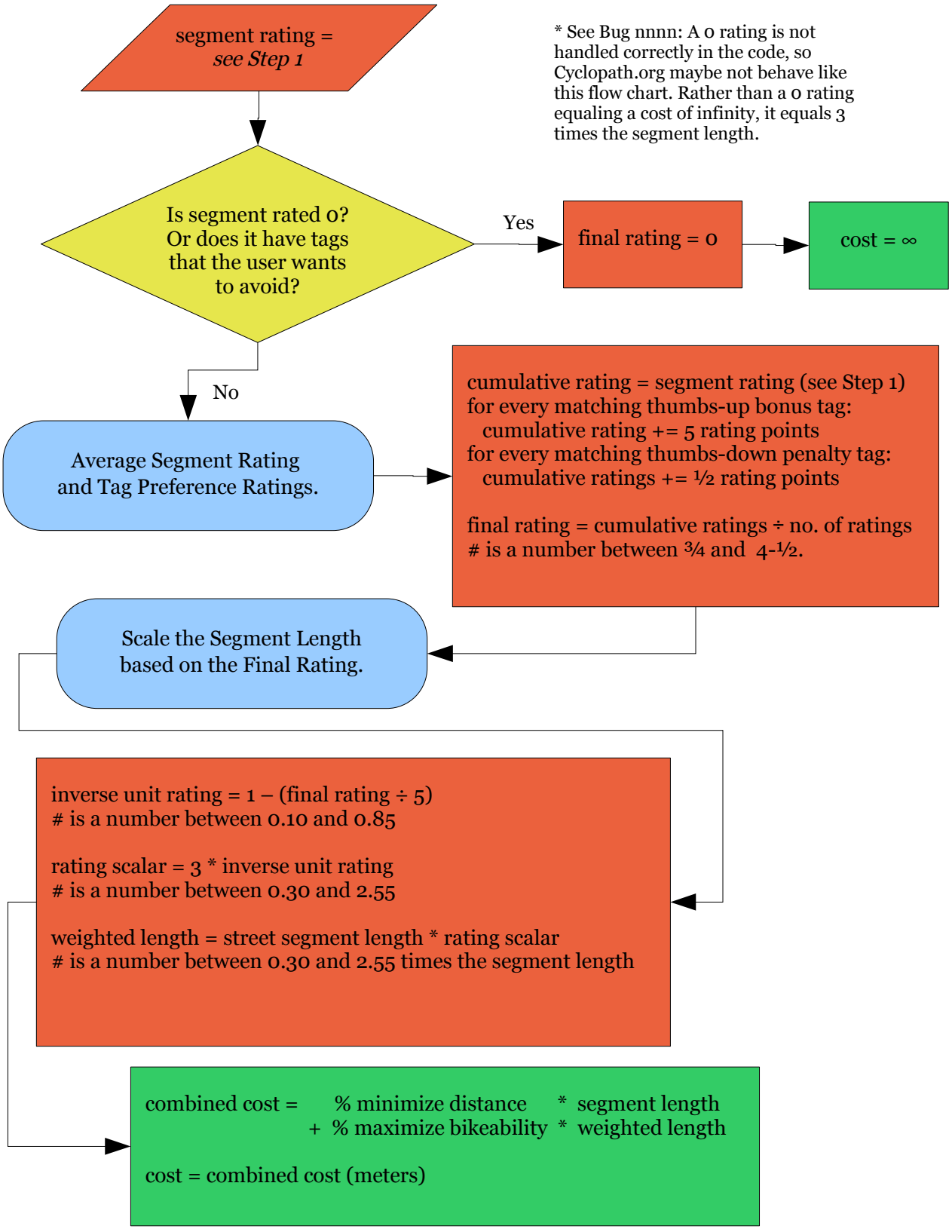
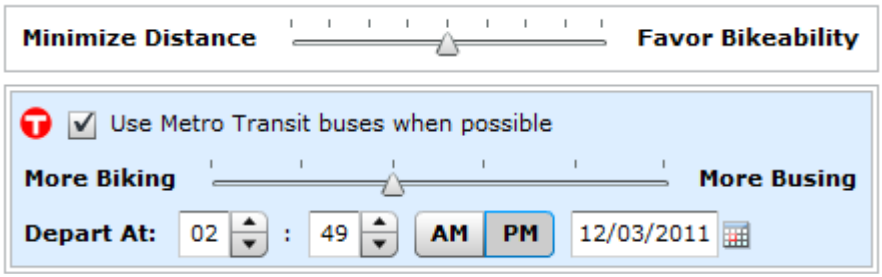


Figure 2. Cyclopath V1 Edge Traversal Cost Flow Chart (Part II).

Cyclopath Route Finder Cost Functions, V2

FROM THE USER'S PERSPECTIVE

The user interface that lets the use influence the multimodal route finder includes a slider control similar to the one used for bikeability.



The Multimodal slider uses two extremes, More Biking and More Busing.

Unlike the Bikeability slider, which has nine ticks, the Multimodal slider has just six ticks. Furthermore, the Multimodal slider ticks do not represent a ratio between the two extremes. I.e., the ticks in the Multimodal slider do not represent the percentages, 0, 20, 40, 60, 80, and 100.

This is because GraphServer, which is a multimodal route finder, has to consider two different road networks – Cyclopath's road network and the public transit network – whose nodes only connect at certain times of the day. For example, you can bike to a bus stop whenever, but you can only get on the bus when it arrives (and before it departs).

There are also a few other unique problems with multimodal route finding besides only being able to transition between networks at certain times. There's the issue of how long someone is willing to wait at a transit stop until boarding. There's also the issue of how far someone is willing (or desiring) to walk or bicycle before boarding a transit vehicle (or perhaps after

alighting from one).

TICK MARKS

The Multimodal slider assigns each tick mark a set of values that attempts to capture the different desires of different users.

(FIXME: This is a poor control to use, since the user probably sees the difference between each tick mark as being equal in magnitude, which is not the case. Also, the user interface does not explain what variables the slider influences. And introducing more elements into the user interface will only increase its complexity. Perhaps identifying the most likely "types" of users, giving them descriptive names, and using a drop-down list is the right approach.)

The Multimodal slider's tick marks correspond to the following sets of values. We'll discuss the meanings of the values later.

Tick Position	Bike Reluctance[5]	Maximum Bike Dist.	Overage Penalty[6]
1st	0.50	1,000 km.	0.0
2nd	0.75	1,000 km.	0.0
3rd[7]	1.00	1,000 km.	0.0
4th	1.00	1-1/3 x straight-line distance	0.1
5th	2.00	2/3 x straight-line distance	0.1
6th	2.00	0.0	0.1

5 In the GraphServer source code, this is called "Walk Reluctance." We could just as easily call it "Non-transit Reluctance." But please don't call it "Exercise Reluctance."

6 Overage: Don't go negative, unless you want a century-long spaghetti ride. It means the further from the start you are, the more favored the edge.

7 The third tick is meant to be the "default" setting. It's not suppose to favor either transit or walking/biking/non-transiting.

MULTIMODAL THE ALGORITHM

Cyclopath uses GraphServer to perform multimodal route-finding, but Cyclopath overrides the builtin GraphServer cost function for bikeable edges (which isn't to say we didn't get a lot of great ideas from the GraphServer source code =).

Cyclopath's multimodal cost function for traversing bikeable edges first considers the edge rating, which is generated in the same fashion as in V1: If the user is logged in and has rated the street segment, the user's rating is used. Otherwise, the average of all user ratings is used, or, if unrated, the generic rating is used.

If the segment rating is below a particular threshold – less than ½ rating points – the edge cost is considered infinite, or impassable (which, in the case of GraphServer is just a Very Large Number – 1,000,000 meters). This applies to segments whose road classification is controlled-access, those that are tagged 'closed' or 'restricted', and those the user has rated “Impassable”.

If the segment is not rated below the threshold, we calculate its traversal cost.

I. Average Biking Speed

First, we need to convert the street segment length from meters into seconds. This is so we can relate the road network and the transit network. To calculate how long it takes a cyclist to bicycle the length of a street segment, we need to know the length of the segment and the velocity of the traveler. We can also use the slope or grade changes along the segment to make a more intelligent guess about the velocity.

i. Roadway Slope

The slope of the segment is calculated from the rise and the fall of the street. The rise is the number of meters of elevation as you travel the line segment, and the fall is the number of meters of descending. Since Cyclopath only stores the elevation of street segments at their

endpoints, we only know either the rise or the fall, and this value might be less than its true value (e.g., if the street segment rises to a crest and then falls afterward).

> slope = number of meters of elevation change ÷ length of street segment in meters

To make the following discussion clear, let's do a little trigonometry refresher[8].

> The slope is unit-less and indicates a ratio. If you'd like to think of it in terms of a angle, you can use the tangent function: where m represents the slope and θ is the angle a line makes with the horizontal axis, $m = \tan \theta$. Or, $= \theta \arctan m$. A horizontal line has m = 0. A larger slope indicates a steeper line. And a 45° hill has a slope of 1 – which is quite ridiculously steep for a roadway. Traffic engineers often refer to slope as a percent – so a slope or grade of .01 is a 1% gradient. The larger slopes you'll find in the world range from 20 to 30%[9], or 0.2 to 0.3 (and maybe higher: Filbert Street in San Francisco has a 31.5% slope, or 0.315)[10]. Lastly, slope depends upon direction of travel: downhills have a negative slope, and "upgrades" have a positive slope.

(FIXME: Cyclopath could be coded to know about slope changes along a segment. Also, the elevation data is not entirely accurate, e.g., the elevation of bridges reflects the earth below the roadway. We would need a user interface control to let users correct this information.)

> The algorithm begins with the assumption of an average bike speed of 4.5 meters per second, or about 10 mph, over flat ground.

FIXME: The user interface does not let the user enter the average biking speed. The user could be allowed to enter their anticipated average travel speed, either directly, as an ordinal, or indirectly, by choosing from one of three classifications generally used to describe a cyclist's riding preference: whether they're an A-, B-, or C-rider (something like 12ish, 15ish, and 18ish mph average rides).

---

8 https://en.wikipedia.org/wiki/Slope

9 https://en.wikipedia.org/wiki/Grade_%28slope%29

10 http://www.aviewoncities.com/sf/lombardstreet.htm

FIXME: Should the algorithm consider accelerating and decelerating? Or is this information captured by the node state changes?

For rises or falls, the average biking speed over flat ground is modified according to the slope.

ii. Downhill Speed

For a descending slope, the expected velocity is determined by the “downhill fastness”.

The downhill speed is

downhill fastness * grade + flat ground biking speed

Where

downhill fastness = -12.1

FIXME: insert table showing different downhill fastnesses (?, ?, ?), grades (0.01, .02, .04, .06, 0.1, 0.15, 0.2, 0.3, 0.4), biking speeds (a, b, c)

iii. Uphill Speed

For upgrades, GraphServer uses a bit of magic, employing one of either two calculations depending on the uphill grade.

a. For Slight Inclines

For subtle inclines, the author of GraphServer has indicated that cyclists do not actually slow down but instead gradually speed up (FIXME: Is there research to back this up?). As described in the GraphServer source code, “An interesting thing happens at a particular grade, when they [the cyclist] settle in for a long slog."

GraphServer calls the angle at which the "long slog" becomes a true hill (and slows the cyclist down) as the "phase change grade." The "velocity between 0 grade and the phase change grade is Ax^2+Bx+C, where A is the phase change velocity factor, B is the downhill fastness, and C is the average speed [and x is the grade]."

The phase change velocity factor is based on the phase change speed, which is the average speed during the "sweet spot."

The phase change speed is

$$\frac{\text{uphill slowness} * \text{flat ground biking speed}}{\text{uphill slowness} + \text{phase change grade}}$$

Where

$$\text{uphill slowness} = 0.05$$

and

$$\text{phase change grade} = 0.045$$

The phase change velocity factor is

$$\frac{\text{phase change speed} - (\text{downhill fastness} * \text{phase change grade}) - \text{flat ground speed}}{(\text{phase change grade})^2}$$

(FIXME: Is there an easy way to describe exactly what this values represents?)

The sweet spot speed is

$$\text{phase change velocity factor} * (\text{grade})^2$$

+ downhill fastness * grade
+ flat ground speed

b. For Steeper Inclines

For inclines greater than the phase grade change (4.5%), we calculate a value like the phase change speed, except that we use the actual grade.

The upgrade speed is

$$\frac{\text{uphill slowness} * \text{flat ground biking speed}}{\text{uphill slowness} + \text{grade}}$$

iv. Sample Values

FIXME: Table of sample values to get an idea what this all means! =)

II. Raw Traversal Time

Using the grade average speed, computed in the last step, we calculate the time it takes to ride the edge.

raw time = street segment length ÷ grade average speed

III. Intersection Penalty (Node Traversal Cost)

We penalize the rider for turning or crossing an intersection. Since our cost value is now expressed in units of time (seconds), the intersection penalty is also expressed in seconds. The penalty values are based on the V1 values, which are expressed in meters; we made the conversion assuming a 12.5 mph velocity.

The penalty is calculated now – during edge cost calculation – because we have to know which

direction the cyclist turned to reach this edge. And also because GraphServer doesn't have a hook for other cost functions (we can only influence the edge cost function, unless we have the GraphServer source code). So the cost we calculate for each edge also includes the cost of crossing from the previous edge across the node to the edge.

FIXME: See the discussion above about the node cost. The V2 algorithm is merely mimicking the V1 algorithm. Ideally, we should have known costs for every combination of edge-node-edge. I.e., is it a four-way stop, a 2-way, a traffic control, nothing, etc.

Left Turn – 22.2 secs. (100 meters / 4.5 m/s (10 mph))
Forward – 4.44 secs. (20 meters @ 10 mph)
Right Turn – 8.88 secs. (40 meters @ 10 mph)

IV. User Rating

As discussed above, the user rating is a number from 0 to 5, where 2.5 is an 'average' rating (FIXME: Or is it 3?), 0 means to avoid the segment, and 5 means the segment is ideal for biking. The algorithm short-circuits earlier if the rating is less than the minimum rating, so at this point in this calculation, the rating is between ½ and 5.

We divide the rating by 5 and subtract from 1 to normalize it – this gives us a value between 0.0 and 1.0 that represents the “weight” or influence of the rating. Since 5 was a good rating, and 1 – 5/5 = 1 – 1 = 0, we know that a lower weight indicates a better rating.

bikeability = (1.0 – (street segment rating / 5.0)) # 0.0 to 1.0

If the rating is “positive”, i.e., greater than 2.5, we scale it to have less of an influence than if the rating is “negative”, or less than 2.5. The problem is that you can't reward edges too much, or else the GraphServer algorithm will never consider using the transit network. The costs of the Cyclopath edges, which are expressed in seconds, need to somewhat relate to the transit edge costs. FIXME: Can we do two searches? Search first based solely on travel time so we can figure out where the transit nodes are, and then search a second time using ratings to find the best

bike portions of the route?

If the rating is better than average (> 2.5), we make the bikeability (which is 0.0 to 0.5) into a weight of 0.9 to 1.0.

bikeability scale = (0.9 + (0.1 * (bikeability * 2.0)))

But if the rating is average or less (<= 2.5), we make the bikeability (which is 0.5 to 1.0) into a scalar between 1.0 and 1.2.

bikeability scale = (1.0 + (0.2 * ((bikeability - 0.5) * (2.0))))

We know have a bikeability weight, based on the street segment rating, which is valued between 0.9 and 1.2.

The bikeability is modified one final time based on the Bikeability slider, which indicates a value between 0.0 (Minimize Distance) and 1.0 (Maximize Bikeability).

FIXME: This calculation has a bug. Please see Bug nnnn (delta_w *= bikeability_weight * bikeability_scale).

V. Reluctances

The edge cost is rewarded or penalized based on the biking reluctance. See the table above for which values correspond to which tick marks. The biking reluctance is generally one of the value, 0.5, 0.75, 1.0, or 2.0.

delta_w *= biking reluctance

GraphServer defines a hill reluctance. Which Cyclopath thinks is silly. Bikers *like* hills, am I right?

As of 2011.06.28, hill reluctance is 0.0, this this does nothing:

```
delta_w += rise * hill reluctance
```

VI. Overage Penalty

The overage penalty is used to try to force a long ride to eventually use transit.

According to the Multimodal slider value sets described earlier, the overage penalty for More Biking requests is considered infinite. For a half-and-half request, the overage penalty starts at 1-and-1/3 the straight-line distance between the source and destination nodes. For an almost all More Busing request, the overage penalty starts at 2/3 the straight-line distance. And for a More Busing request, the overage penalty begins immediately, at 0 meters. The overage factor is always 0.1 for Cyclopath.

FIXME: Landon got these values from playing around. Maybe more study is needed?

While GraphServer is searching the network, if the total path cost exceeds the overage limit, the edge cost is appended with an overage penalty. The penalty increases the more the path cost is over the limit.

```
delta_w +=  (( distance biked + street segment length - max bike dist)) # excess meters
            * overage factor          # 0.1
            * delta_t)                # secs to trav edge
```

FIXME: I'm not confident about this value: The penalty gets compounded quickly, doesn't it?

VII. Final Cost

GraphServer does one final mutation – multiplying by the “slog” – but Cyclopath skips this step. (Landon, for one, can't really figure out what it is – I think it's a workaround for code that uses the builtin cost function but for which good paths aren't being found.)

So the cost of traversing an edge in the context of multimodal considers a number of elements:

- Edge Slope
- Time to Bike the Edge
- Intersection Penalty
- User Rating Bikeability Weight
- Minimum Distance Reluctance
- Maximum Distance Overage Penalty

GRAPHSERVER TRANSFER PENALTY

The GraphServer search algorithm lets you assign a penalty, in seconds, for each boarding. If this value is too low, the search may include frivolous transfers. If the value is too high, the search may avoid the transit network entirely.

transfer penalty = 5,000 (FIXME: Is this 5,000 msec./5 secs. or 5,000 secs./83 min?)

FIXME: Look at the GraphServer source and understand this better. I.e., when does the penalty apply to the path?
FIXME: See also what GraphServer's walk option's num_transfers does.
FIXME: What's the alight penalty?
FIXME: What's slog?
FIXME: What about reverse_of_source or arrive-by searching?

GRAPHSERVER INNER WORKINGS

FIXME: Please skip this section. I want to document the types of edges and nodes in the transit network, which I think will help explain the interaction between the two networks and help illustrate how the GraphServer search algorithm works.

Following is a note about the difference between PSV and STA nodes, from Brandon Martin-

Anderson (Graphserver Hero Extraordinaire). FIXME: Explain this better.

> "sta-" vertices are "station" vertices and correspond to physical transit stops. "psv-" are Pattern-Stop Vertices, or PSVs. These vertices represent the state of being on a transit vehicle traveling on a particular pattern at a particular stop. For example, a class of edges called "TripBoard" edges model moving between station vertices which model being at a station and _off_ a vehicle to pattern-stop vertices which model being at a station _on_ a vehicle. Then a class of edges called "Crossing" edges model going between to PSVs. It's a little contrived, but it's necessary to work around the dreaded Pro-Active Transferring Bug.[11][12]

11 https://groups.google.com/group/graphserver/msg/7a18e62fdccf0722

12 https://github.com/bmander/graphserver/wiki/Board---Alight---Crossing-Transit-Graphs